Database Project Phase II

Authors: Dorsa Abdi, Taha HosseinPour, Parinaz Kanan, Nazanin Zarei (alphabetical order)

Professor: Professor Shirazi

Course: Database

Term: Winter-Spring 2024

Overview

During this phase, our team was tasked with creating a dynamic dashboard/interface to read and write data from/in our database. The dashboard is intended to be utilized by the restaurant's manager and/or admin assigned to the job.

Design Decisions

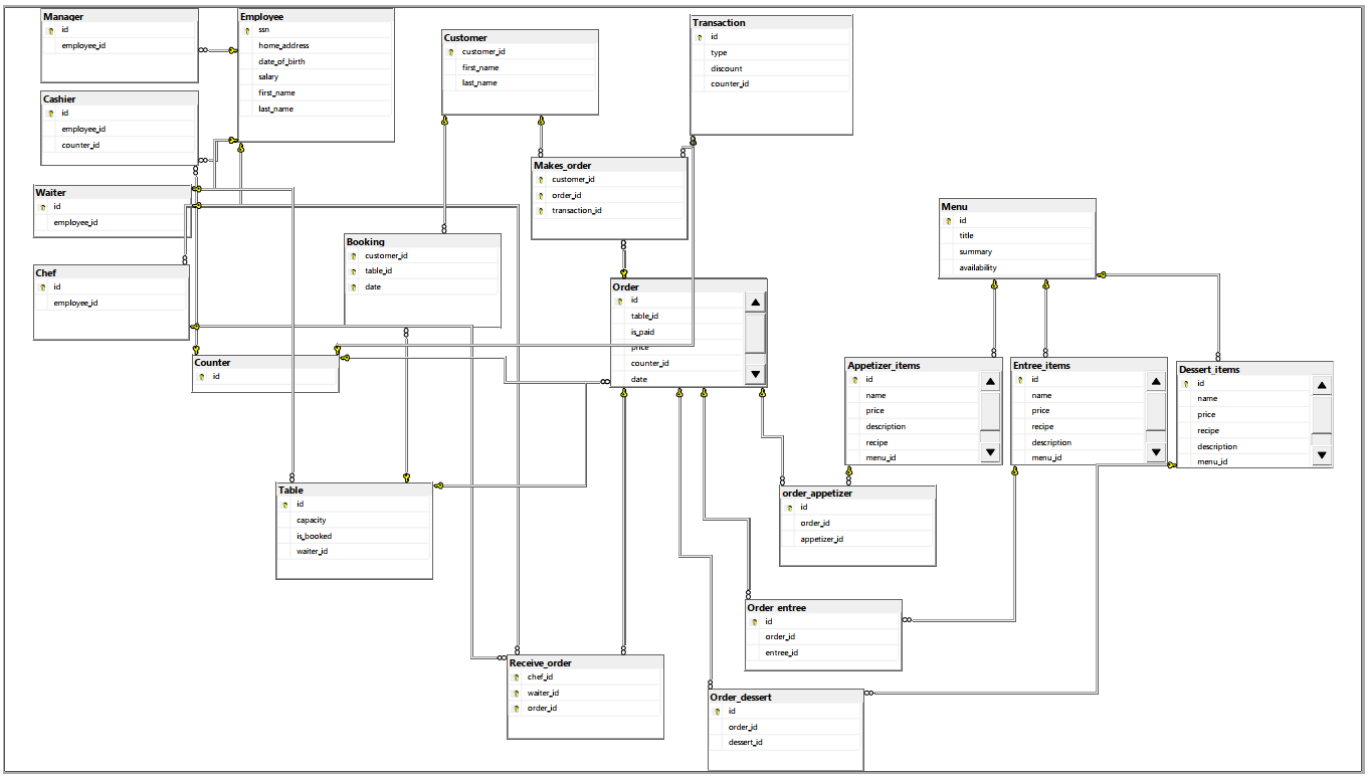
1. The dashboard is structured based on the action the admin will choose: Insert, Delete, Update, View.
2. In the insert page, the admin can select the table to insert the data into. If the ID of that table already exists, it will not insert the information.
3. In the update page, the admin can choose the table to update the data in. If the ID of that table does not exist, it will not take any action.
4. In the delete page, the admin can choose the table to delete the data from. If the ID of that table does not exist, it will not delete anything.
5. The admin can check the status of every table on the view page.
6. There's also a section on the view page for running more complex queries and viewing the results.

Specifications and Requirements

- User-friendly front-end designed using Python's Streamlit library along with HTML and CSS.
- Back-end implemented using Python's Streamlit library.
- Sustainable connection to the database ensured.
- Insertion, deletion, and updating of data facilitated through respective pages in the dashboard.
- Testing conducted to verify functionality.

Database Schema

The database schema is available in the repository alongside all the code for this phase of the project. The project is implemented using SQL Server, and a MySQL schema picture is provided for better understanding.



Instructions

1. Clone the repository using:

```
git clone https://github.com/ewondare/restaurant_db
```

2. Make sure to replace the server and database name in the connection string with your own local server name and database name in each of the code file you find in the pages folder.

```
DRIVER_NAME = 'SQL SERVER'
SERVER_NAME = None # YOUR SERVER NAME
DATABASE_NAME = None # YOUR DB NAME
```

3. To find your local server name, open SQL Server Management Studio and run the following query:

```
USE your_database_name;
GO
SELECT @@SERVERNAME;
```

4. Open the current directory in a Windows PowerShell or activate your virtual environment (if used).
5. Change directory to the folder containing `dashboard.py`.
6. Activate the dashboard using the command:

```
streamlit run dashboard.py
```

7. After a few seconds, a new window of your chosen browser will open with the dashboard deployed locally. From there, you can use all the commands to manipulate data in the given database.

`dashboard.py`

Overview: `dashboard.py` serves as the main interface for the Restaurant Access Dashboard project. It provides a user-friendly web application for administrators to manage restaurant data efficiently.

Functionality:

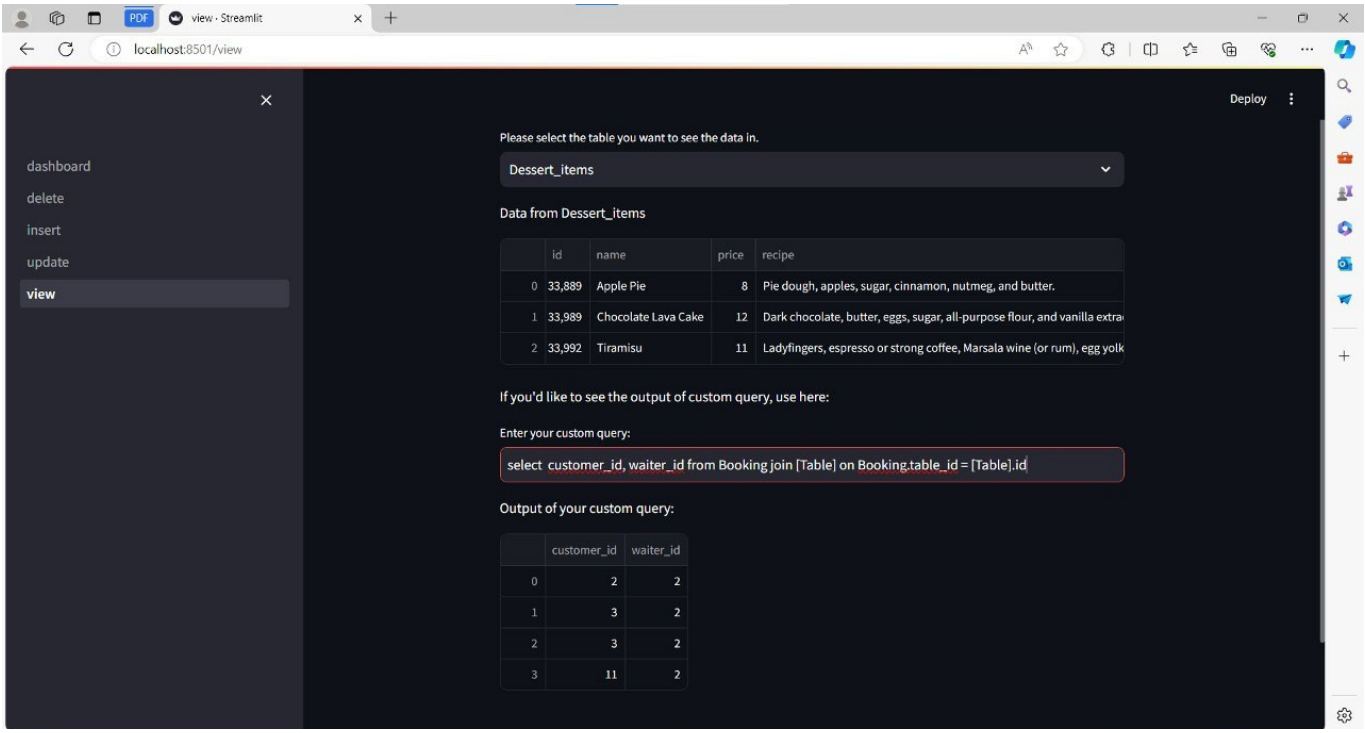
- Implements a sidebar menu for easy navigation.
- Configures page settings such as title and layout.
- Utilizes Streamlit for dynamic web application development.

Technical Details:

- The script utilizes Streamlit, a Python library, for building interactive web applications.
- It sets page configuration options using the `st.set_page_config()` method, allowing customization of page title, icon, and layout.
- The sidebar is created using `st.sidebar`, providing a convenient navigation menu for users.

`view.py`

Overview: `view.py` enables administrators to view and analyze restaurant data stored in the database. It offers flexibility in selecting tables and executing custom queries.



Functionality:

- Establishes a connection to the SQL Server database.
- Allows users to select tables for data visualization.
- Supports custom SQL queries for tailored data analysis.

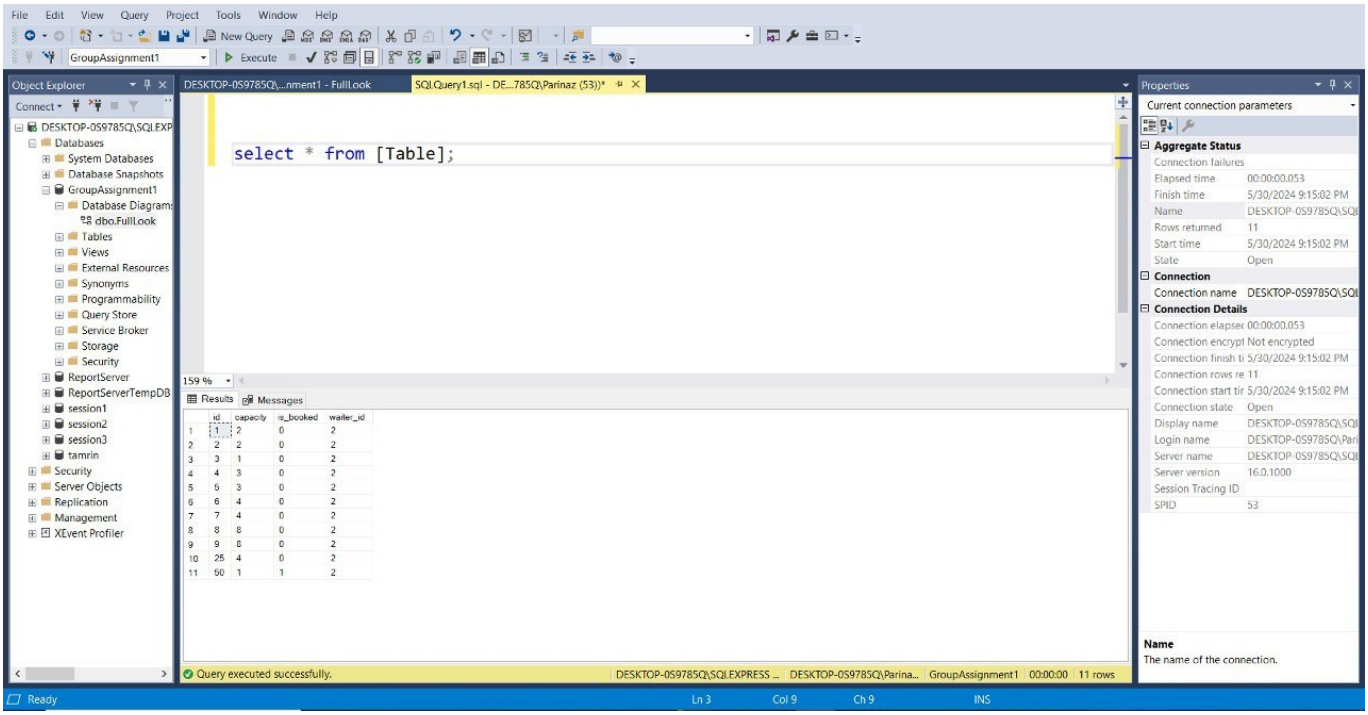
Technical Details:

- The script establishes a connection to the SQL Server database using the `pypyodbc` library.
- It dynamically generates table selection options based on the available tables in the database.
- Custom SQL queries are executed using parameterized queries to prevent SQL injection vulnerabilities.

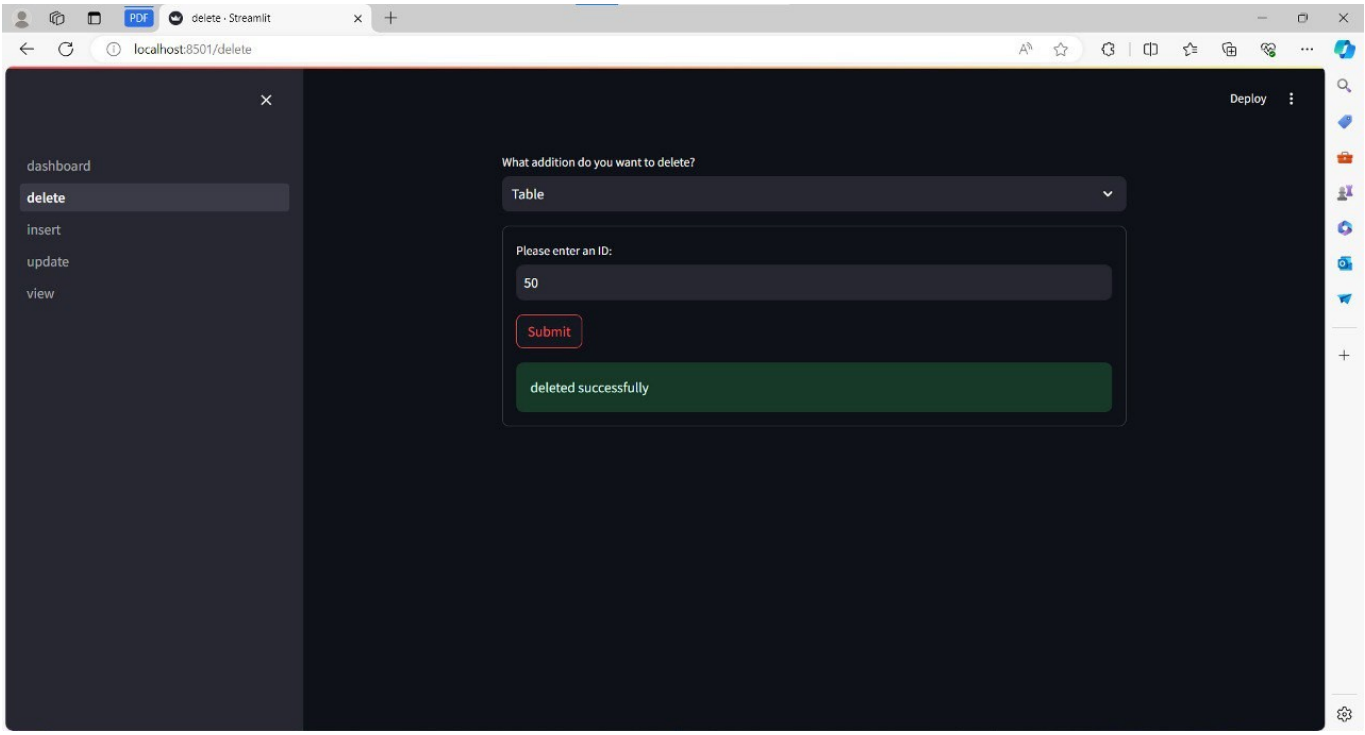
`delete.py`

Overview: `delete.py` facilitates the deletion of records from the database. It provides options for targeted deletion while ensuring data integrity.

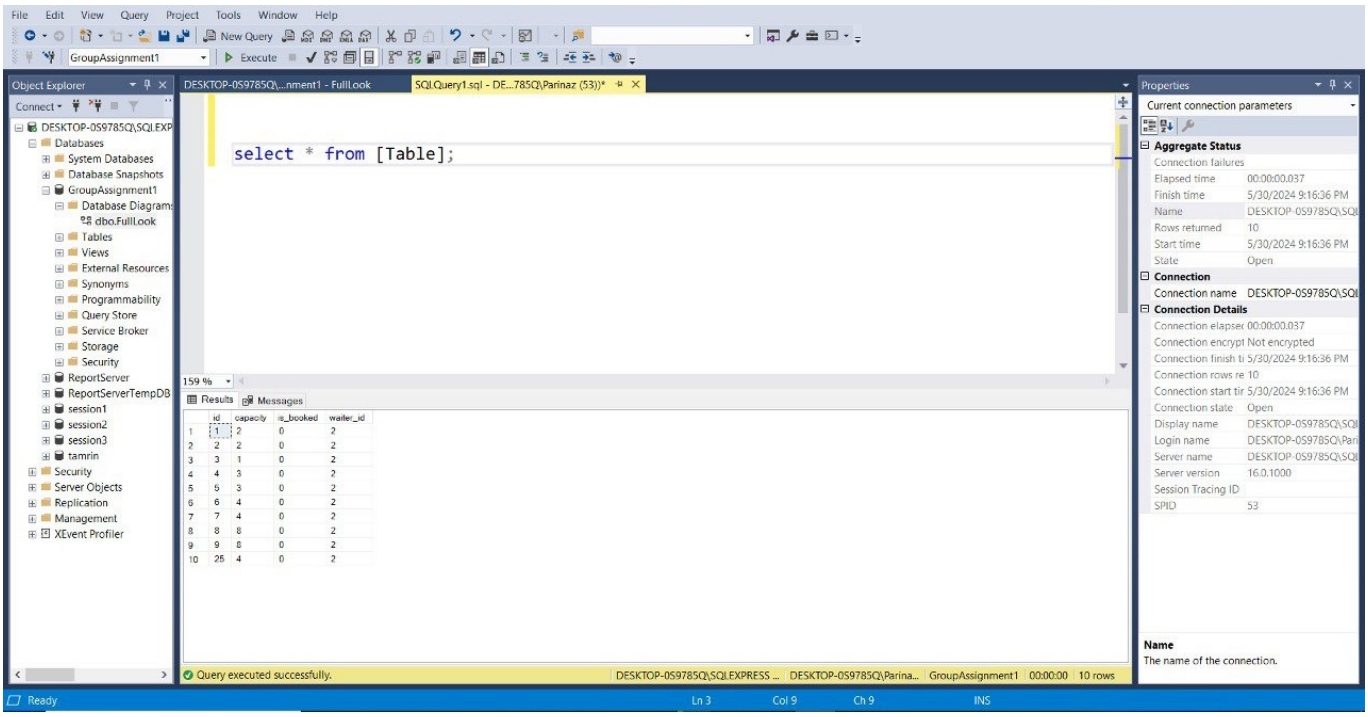
Before deleting:



Delete page:



After deleting:



Functionality:

- Offers table selection for precise record deletion.
- Implements form submission buttons for user interaction.
- Includes error handling mechanisms for graceful error management.

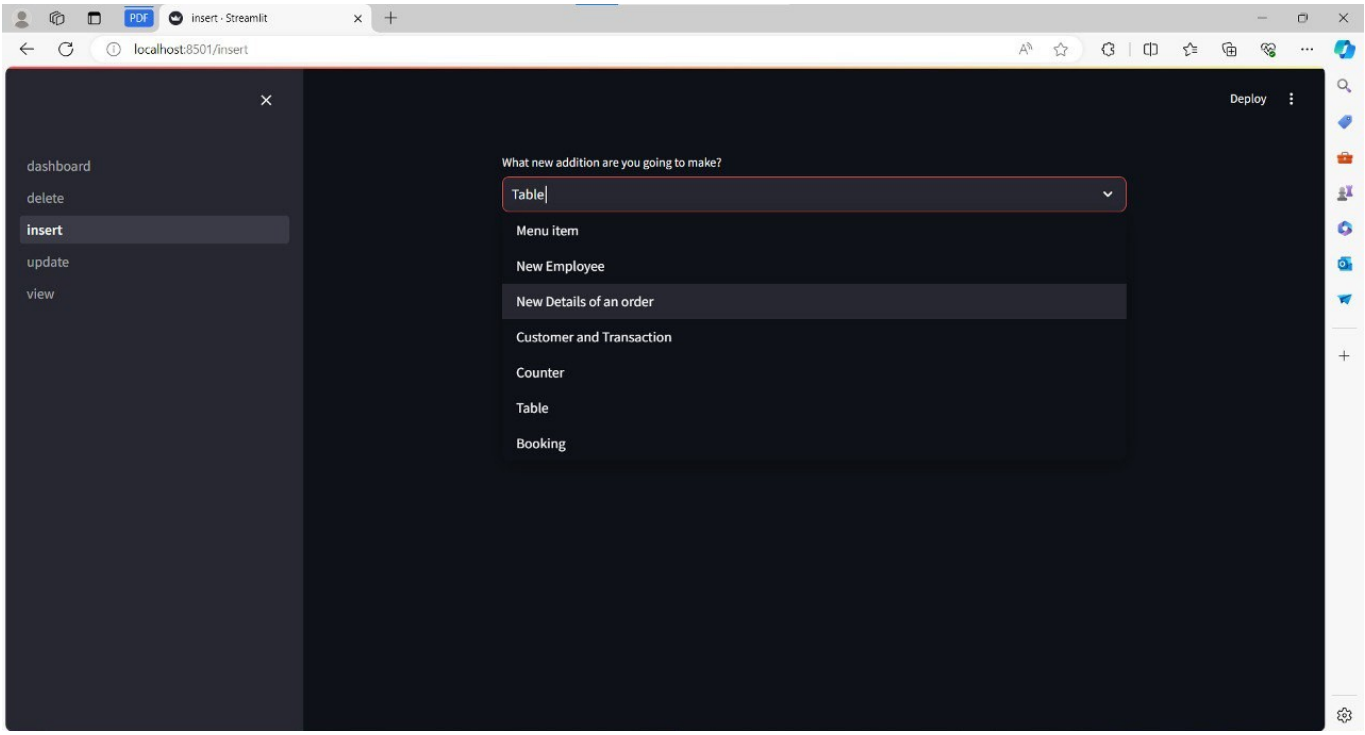
Technical Details:

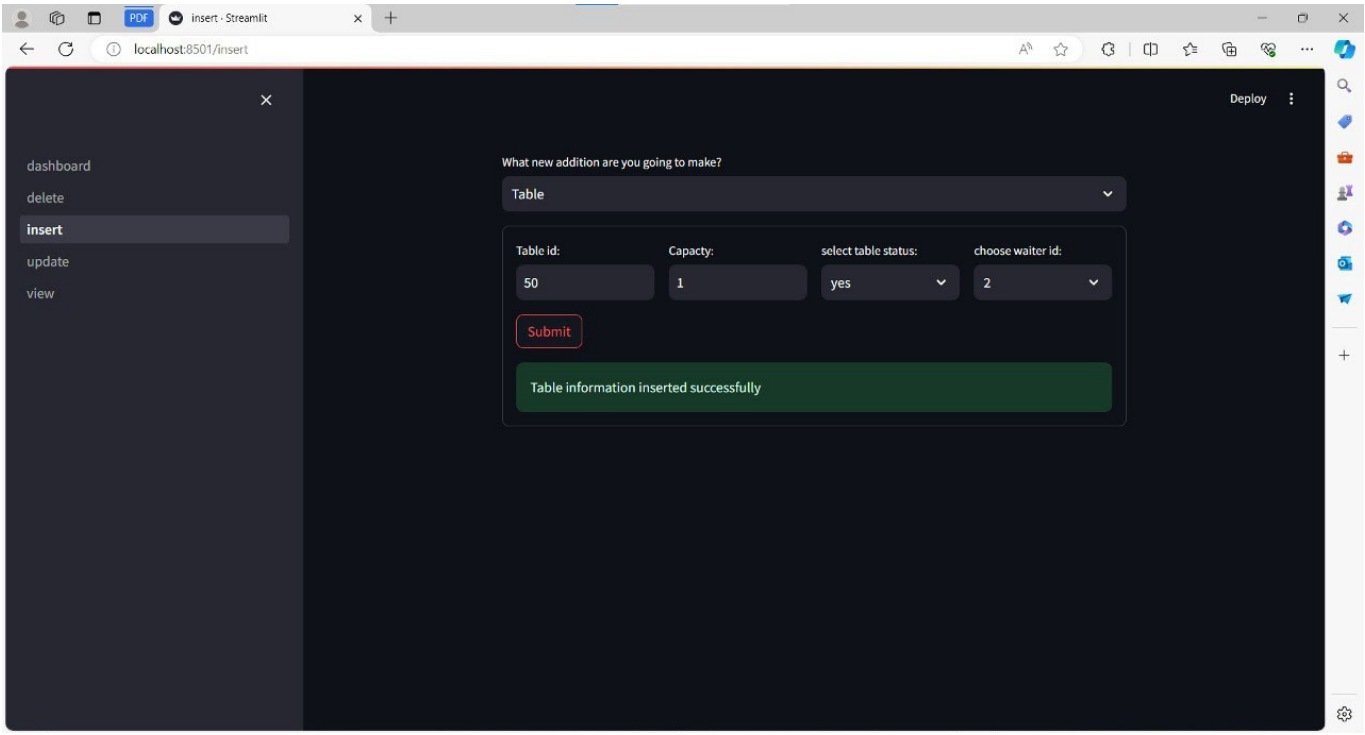
- The script connects to the database using `pypyodbc` and provides options for selecting tables.
- It uses form submission buttons to trigger delete actions, ensuring user-friendly interaction.

- Error handling mechanisms are implemented to catch and display errors, ensuring a smooth user experience.

`insert.py`

Overview: `insert.py` simplifies the process of adding new data entries to the database. It provides a user-friendly interface for inserting menu items, customer details, etc.





Functionality:

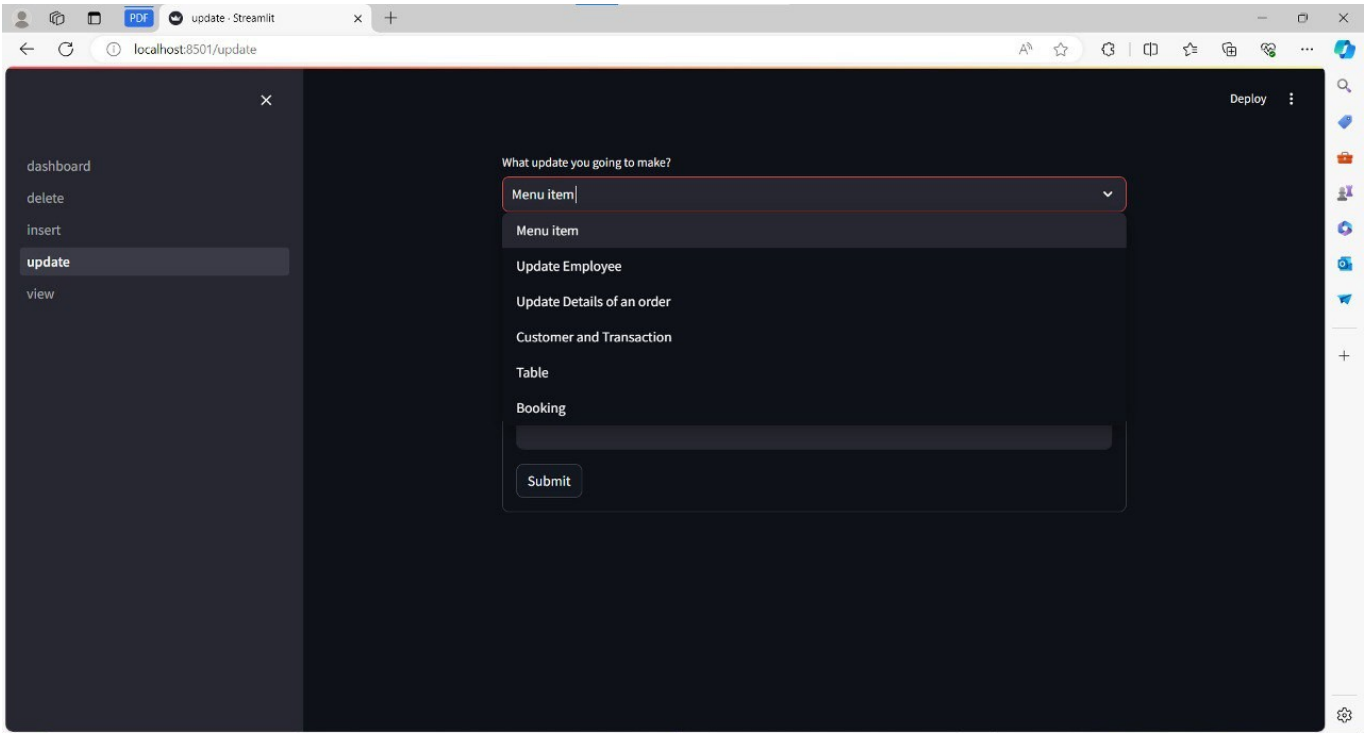
- Validates user input to maintain data integrity.
- Utilizes parameterized SQL queries for security.
- Ensures transactional consistency during data insertion.

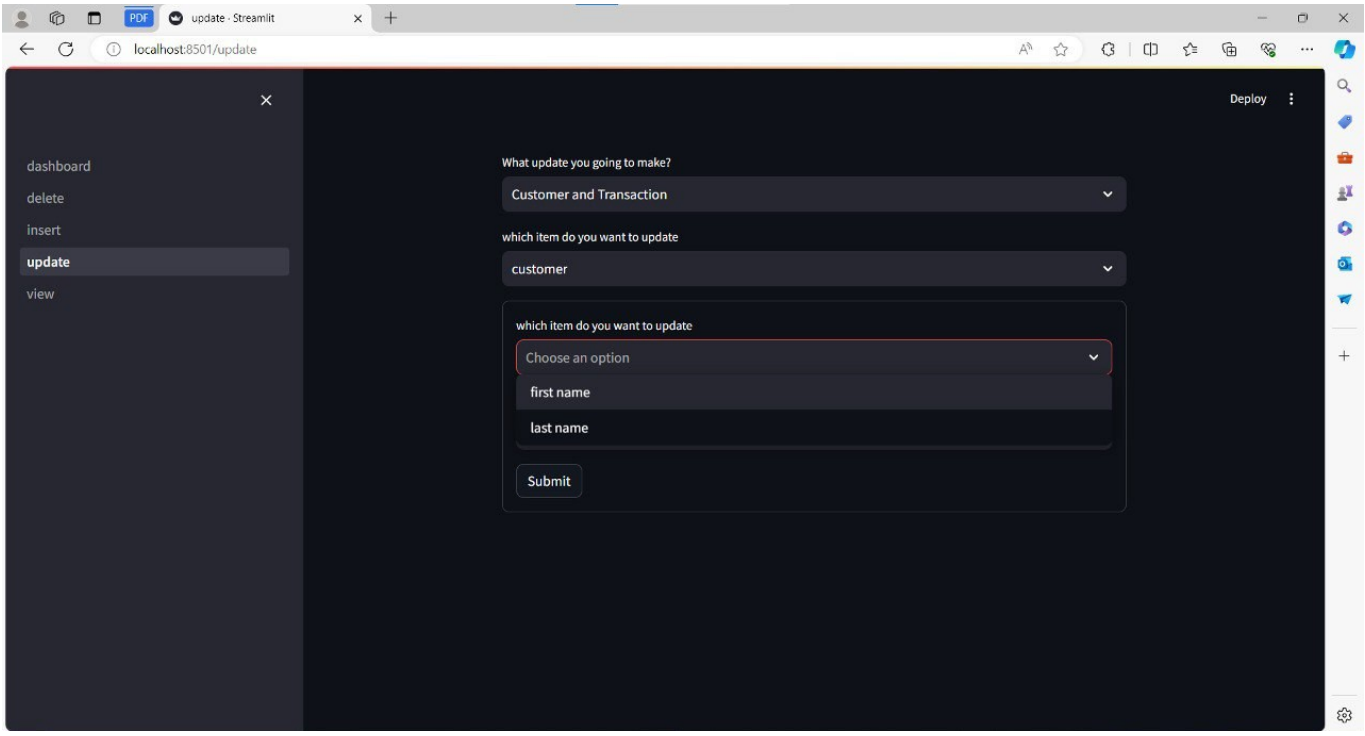
Technical Details:

- The script validates user input to ensure that all required fields are filled out before insertion.
- Parameterized SQL queries are used to prevent SQL injection attacks and ensure data security.
- Transactional integrity is maintained to ensure that data insertion operations are atomic and consistent.

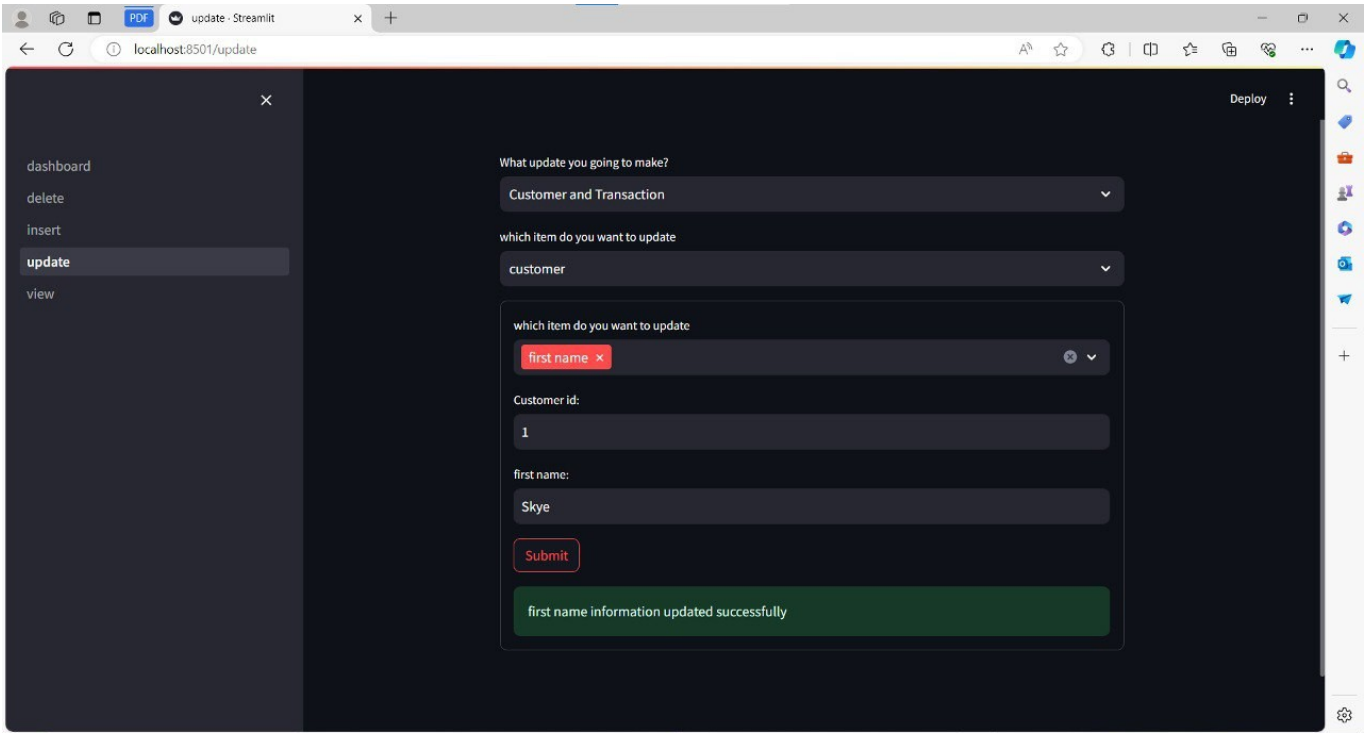
`update.py`

Overview: `update.py` enables administrators to modify existing data within the database. It offers selective updates for precise data modifications.





You can clearly see that our database works dynamically by comparing these 2 snapshots



Functionality:

- Allows selective updates of specific fields within records.
- Implements error handling for error management.
- Ensures transactional integrity during data updates.

Technical Details:

- The script provides options for selecting records and fields to update, enhancing user control.
- Error handling mechanisms are implemented to catch and display errors, ensuring a smooth user experience.
- Transactional integrity is maintained during data updates to ensure that changes are applied atomically and consistently.

Thanks for your interest in our project, don't forget to start our repository if you found it helpful 😃